SILVERSOFT

Introducing the ZX Compiler

Instructions on loading.
Place the cassette on side A, into the recorder. Follow loading instructions, as found in chapter 16 of the ZX81 manual. If at first the program does not load, try different tone and volume levels. If after this it still does not load, try a different tape recorder. There are two copies of ZX COMPILER on the tape. (Use file name" COMPILER").

About the ZX COMPILER
The ZX COMPILER is a program, written in basic and machine code that will change a program in 'BASIC' into machine code. This can be used to create programs or sub routines that can run up to twenty or thirty times faster than normal 'BASIC'. Because the ZX81 only has a capacity of 16K ram the compiler works in an integer-only, sub-set of BASIC. To save even more space commands have been left out that are not needed, for example REM:- which is not needed in compiled programs any way and INPUT:- which can be made by a routine using gets and if.....then statements.
Once your program to be compiled is written, you run the compiler. After a period of fast mode, the program 'NEWS' itself and leaves you with a single rem statement containing the machine code (it looks like random junk).
Once written you may either call the code from within a program or directly from the keyboard using LET X = USR 16595.

INSTRUCTIONS ON USE
The program to be compiled is written in 'REM' statements before line 1000 beyond line 1000 is the compiler. The reason for REM's is so you can have extra commands that can't be keyed-in, in normal BASIC. This does mean though that you will have to key in whole commands for example PRINT is P-R-I-N-T. eg:

```
1 REM PRINT "HELLO"
```

In the program there are to be no spaces, except in print statements. Also goto statements must be followed by a 'true' line, for example the following will not work.

```
1 REM PRINT "HELLO"
2 REM GOTO28
```

When you type run and newline, the screen will blank for a few seconds and then print up:
ERROR IN LINE 28
Which means it can't find any line 28
The compiler program may only have variables A-Z, no strings or arrays. (These may be implimented using other commands such as peek and poke).
-variables in the compiler are not the same ZX81 variables.

Numbers may be 0-65535. To get a negative number use 65535-x eg: -2=65536-2
There may seem a lot you can't do but it is enough to write most programs, (such as space invaders, breakout, etc.) The only way to really find out what the compiler can do is use it, read through the examples and work them out.

COMPILER COMMANDS

In this section I will go through all the commands available on the compiler.
In this section (variable) can mean a number of things:

1) A number 0-65535
2) A letter A-Z
3) RND followed by a variable
4) USR followed by a variable
5) TOP
6) DEEK followed by a variable
7) PEEK followed by a variable

eg: RNDA RND200 RNDUSRDEEK16514!!
Top and Deek need explaining, the others should be clear enough. Top will return the first position on the display that can be poked. It's basic equivalant is : PEEK 16396 + 256 * PEEK 16397 + 1
Deek is the opposite of doke and peeks two locations at the same time. (To return a number 0-65535) Top is the equivalant of DEEK 16396 + 1. The nascom and other computers use these commands.
eg: It can be used for poking a lazer base with one command.
1 REM DOKE TOP 1153

PRINT

This can either be followed by text enclosed in speech marks or a variable, (A-Z) it will print the chrs of the variable. The semi colon at the end of the print is automatic. To print at the start of a new line use the following:
1 REM PRINT 118
(Which is the same as the BASIC PRINT CHRS 118.) So print can be used in two forms which are:
1. PRINT,"--TEXT--"
2. PRINT (VARIABLE)

GET

This is the same as the BASIC INKEY, except only it returns the code (0-128) of the key pressed. It may only be used in one form, if no key is pressed it returns 128.
1. GET (VARIABLE)
eg: 1 REM GET A 2 REM PRINT A 3 REM GOTO 1

END

This is the same as the BASIC STOP, eg:
1. END
It is important that you end a compiler program with either an END if you want execution to stop, or RETURN (see later) if you want to continue the calling BASIC program.

PAUSE
This may only be used in one form:
1. PAUSE (VARIABLE)
It is a delay, the length of which is determined by the (VARIABLE). The longest delay possible would be 65535, it lasts for 2 seconds!!

CLS
This is the same as BASIC CLS except it clears all 24 lines of the display, not just 22. It has only one form:
1. CLS

MOVE
This is the same as the BASIC PRINT AT, except you may "PRINT AT" all 24 lines down. It takes only one form:
1. MOVE (VARIABLE),(VARIABLE)
eg: 1 REM MOVE 23, 10 2 REM PRINT "HELLO" 3 REM END

LET
This is like the BASIC LET statement except you may not have long equations or brackets. It may take only two forms:
1. LET (VARIABLE) = (VARIABLE)
2. LET (VARIABLE) = (VARIABLE) (OPERATOR) (VARIABLE)
(OPERATOR) can be either +,-,* or /

SCROLL
This is the same as the BASIC SCROLL except that it is a lot quicker and scrolls all 24 lines of the screen.

GOTO
This is like the BASIC goto except the line number must exist. It may only take one form:
1. GOTO NUMBER (1-999)

GOSUB
This is the same as BASIC GOSUB, except for detination line number which must exist. It can only take one form:
1. GOSUB NUMBER (1-999)

RETURN
Used in conjunction with GOSUB, this is identical to its BASIC equivalant. Used when not in a GOSUB sub-routine,it will return you to BASIC and continue with the calling program. It may only take one form:
1. RETURN

PLOT
This is the same as BASIC PLOT. It may take only one form:
1. PLOT (VARIABLE),(VARIABLE)

UNPLOT
This is the same as BASIC UNPLOT. It may only take one form:
1. UNPLOT (VARIABLE),(VARIABLE)

POKE
This is the same as its BASIC equivalant. It may only take one form:
1. POKE (VARIABLE),(VARIABLE)

DOKE

Is used on the nascom and other computers, it pokes two locations at the same time with a number of up to 65535. The low byte of the number at the address and the high byte at address +1. For more information see example programs. It can only take one form:

1. DOKE (VARIABLE),(VARIABLE).

IF is used in only one way:

1. IF (VARIABLE) (SIGN) (VARIABLE)...............

(SIGN) can mean either =,<>,<>. There is no need to use the then statement. For more information see example programs.

eg: IF A>2 GOTO 1Ø

SCROLL

These are all the commands allowed in the compiler. But as you can see from the example programs, these are enough.

EXAMPLE 1

This program will invert the screen, then Return to basic not stop. It will invert all 24 lines of the display.

```
1 REM LETB=24
2 REM LETA=TOP
3 REM IFPEEKA=118GOTO10
4 REM LETZ=PEEKA+128  : NUMBER OF LINES
5 REM POKEA,Z         : TOP OF SCREEN
6 REM LETA=A+1        : END OF LINE ?
7 REM GOTO3           : INVERT IT
10 REM LETB=B-1
11 REM IFB=0RETURN     -"RETURN" TO BASIC, USE END
12 REM GOTO6            STOP WITH AN ERROR CODE'.
```

Try this program out, then try it in BASIC to see how slow it is.

EXAMPLE 2.

This program demonstrates the DEEK and DOKE commands, DOKE is used to 'save' the variable while it is used for something else, DEEK reads it back again.

```
1 REM LETA=45192
2 REM DOKE30000,A
3 REM LETA=0
4 REM LETA=DEEK30000
5 REM IFA<>45192PRINT"THIS WILL NEVER BE PRINTED"
6 REM END
```

In this program location 30000 was chosen because it is high up in the memory and will not interfere with anything.

WHEN YOU HAVE WRITTEN A PROGRAM.

When you have written your program in REM statements, SAVE a copy on cassette (in case you have to de-bug it) Then RUN, this will run the compiler as it will go past REMS. After a delay (10mins for 4K program) the screen will return. If it says error, correct error and re-run it. If it comes up with one rem followed by junk then it has finished compiling, BEFORE YOU DO ANYTHING ELSE type in: 2 RAND USR 16595

Then SAVE another copy of the program and run it. If the program does something odd, try and see what is happening, re-load the first copy and correct the program.

MORE FACTS

Don't forget in machine code there is no 'break' key to test for one use.

1. REM GETA
2. REM IFA= Ø END

Even a one line compiler program when compiled may seem to take up a lot of room, this is quite normal because about 1K is taken up with variables and routines. The compiler was really developed for small fast routines to be used in a BASIC program or entire games in machine code.

If you want to interchange basic variables and compiler variables, the compiler variables are held at 16514 low byte first,high byte next. A(low) is at 16514, B(low) is at16515 and so on.

In the program you will see two sets of inverse AJGs this is so the author can identify any compiled program. (No program may be used personal gain which in any way conected with the compiler without the permission of the author.

Try to use POKE and PEEK, (deeks and dokes) more as they are very fast in the compiler. Things like PRINT; PRINT AT, PLOT, UNPLOT use basic routines to space so they will only be executed at about twice the speed as normal.

The RND function is not very random when displaying the results, but it is good enough for normal random decisions. (This is again due to the lack of space available). Compilers available for other computers may need 4K of code for one line of compiled program!!

MORE EXAMPLES

1) Add all the numbers 1-1000 together.
(returns the answer in s 165114+16515)

```
1 REM LETA=Ø
2 REM LETC=Ø
3 REM LETA=A+C
4 REM LETC=C+1
5 REM IFC  1̄01GOTO3
6 REM END
```

Type this in, then RUN it, to see the answer
PRINT PEEK 16514+256*PEEK 16515
Try out the program in BASIC as well and see the diffrence in speed

2)

```
1 REM LETP=Ø
2 REM MOVE23,P
3 REM PRINT34
4 REM GET A
5 REM IFA=33LETP=P-1        key 5
6 REM IFA=36LETP=P+1        key 8
7 REM IFP=65535LETP=Ø
8 REM IFP=31LETP=3Ø
9 REM SCROLL
10 REM GOTO2
```